DRAFT ACM Computing Surveys v(i), month 1999, http://www.acm.org/surveys/tbd.html.

Intermediary Architecture: Interposing Middleware Object Services between Web Client and Server

Craig Thompson, Paul Pazandak, Venu Vasudevan, Frank Manola, Mark Palmer, Gil Hansen, Tom Bannon
Object Services and Consulting, Inc.
2725 Deep Valley Trail, Plano, TX 75023
thompson@objs.com, http://www.objs.com/bios.htm

Abstract: This paper describes the *Intermediary Architecture*, which interposes a distributed object services architecture between Web client and server. The architecture extends current Web architectures with a new kind of plug-in, making a new collection of Web applications easier to develop. Example services including Web annotations and Web performance monitoring are described.

Categories and Subject Descriptors: H.3.4 **[Information Systems]**: Systems and Software - *Information Networks*

General Terms: World Wide Web, distributed objects, object request brokers

Additional Key Words and Phrases: annotations

1 Introduction

The objective of our project "Scaling Object Services Architectures to the Internet" [Thompson 1998b] is to develop a software infrastructure architecture aimed at unifying Web and distributed object component software architectures. We are working on several aspects of the problem including how to extend the Object Management Group (OMG) Object Management Architecture to accommodate architectural properties like composability, scaling, and evolution [Thompson et. al. 1997; Thompson 1998a], what a web object model might look like [Manola 1998a; Manola 1998b], how to deserialize World Wide Web Consortium (W3C) Extensible Markup Language (XML) documents as Java objects [Pazandak 1998a], and new infrastructures for Web-object middleware integration (the subject of this paper).

Today the Web consists of thin clients, which provide universal browser interfaces, and web servers which provide back-end access to global information resources -- that's where the data is. Meanwhile object request brokers (ORBs like CORBA) provide a distributed object bus for accessing suites of middleware services -- that's where the middleware management services are. At present, ORBs are not really connected well to control access to web data sources. Two approaches to Web-ORB integration are already in common use: backend ORBs wherein server-side CGI scripts call ORB clients that access middleware services; and downloading ORB clients as applets wherein a Java applet is downloaded which contains a Java ORB client that then can communicate to an ORB server. Netscape's Visigenic CORBA client is pre-downloaded to optimize the download time.

This paper describes a third general-purpose Web-ORB integration architecture, which we call an *Intermediary Architecture* (IA). The idea of the Intermediary Architecture is to interpose intermediary middleware plug-in services between the Web client and Web server, thus creating *URL brokers*. One can draw a wrapper-like picture of client-IA-server where the client is red, the server is green, and the IA has a green-red layer so the client sees IA as a server and the server sees IA as a client. This is like mating male and female leads via an adapter. But now, new

• services can be delivered via the intermediary.

The design pattern of interposing intermediary behavior between components is not entirely new. Wrappers do this in an *ad hoc* way. More generally, reflective architectures (e.g., CLOS) provide expansion joints where new behaviors can be interposed. The DARPA Open OODB system [Wells et. al. 1992] used interceptors (called sentries) to install new behaviors like persistence and versioning and treated querying as a service. OMG's security architecture uses fairly general purpose interceptors to install security into OMG's distributed object architecture. The OMG Portable Object Adapter specification provides before-after filters. On the web, most work on intermediaries, has focused on special-purpose proxy servers for page caching mechanisms and security firewalls. Web platform vendors have bundling functionality that could have been added by intermediaries, so thin clients are getting fatter via services like client-side caching, history mechanism, bookmarks, multiple views like Page Source and Page Information, SSL-based security, and versioning. Commercial web clients (like Netscape Navigator) and servers so far do not directly support client side or server side filtering. The experimental W3C Jigsaw server does support server-side filters.

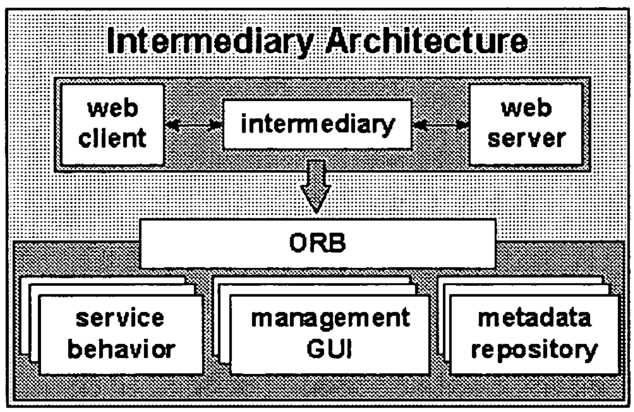

Figure 1: Intermediary Architecture

2 Intermediary Architecture Design Overview

The Intermediary Architecture is shown in Figure 1. The main components of the architecture are:

- **URL interceptor(s)** [Pazandak 1998b] - these enable insertion of side-effect behaviors before or after client send, server receive, server send, or client receive operations. The interceptor infrastructure consists of a *service plug-in API*, which provides a means of registering and invoking services at run time including the ability to add, remove or turn on-off services, and *service composition specifications*, which sequence a collection of services so they are executed in a known order.
- **intermediary services** - these implement some behavior that a third party can specify, possibly remotely. Services can be implemented in Java or C++ or by using distributed programming architectures such as CORBA or ActiveX.
- **service support infrastructure** - services may share a common service support infrastructure consisting of:
  - **a system management policy** - governs a service's behavior.
  - **metadata repository** - services typically store their *metadata* in a (local or remote, possibly shared) *repository*.
  - **trader** [Vasudevan 1998; Vasudevan and Bannon 1998] - can be used to discover and federate service interface repositories and service metadata repositories for scaling.
  - **security infrastructure** - needed to allow trusted third parties to install and maintain support services.
  - **management GUI** [Hansen 1998b] - needed to toggle service policies and parameters

The architecture can appear in many variations:

- Many kinds of services can be inserted via an intermediary, including security, versioning, internationalization, query augmentors, shielded pages, rating systems, rerouting, filtering, logging, system management instrumentation, clocking, micropayments, disconnected and intermittent access, tracking URL usage patterns of a community of interest, background indexing of pages you have seen before so you can find them again, indexing streams of audio or video, translingual translation service, closed caption in multiple languages, and augmenting a web page with voice access to URLs.
- There can be more than one intermediary. Different groups in an organization could manage security, caching, filtering, system management and logging, functions like annotation, the ability to add new functions, etc.
- Services can be inserted dynamically at run-time. One way to do this is via a *Trader* that provides service discovery. The trader can do this once at the beginning or can be continuously polling to locate new or better service providers or to make the system more robust if some providers fail.
- Intermediaries can be implemented as adjunct to the client or the server or somewhere in between. We talk about client intermediaries, server intermediaries, and proxy intermediaries.
- A given service can store metadata it collects in a repository. If local to the Web client, the service might provide *personal services* (e.g., personal Web annotations -- see the Annotation Service below). If shared in a group, the service can provide *group services* (e.g., group annotations). If shared within a community or on the server it can provide *public or community services*.
- Repositories can be federated together to allow more sharing.
- Repositories can be different for different services or they can be shared across services, e.g., so that both the performance monitor and the annotations services could share one repository.
- It is important to be able to view and query the metadata and adjust the policy associated with a service.
- Federation can appear in many places within this architecture: individual services can be federated; repositories can be federated.

3 Intermediary Architecture Examples

To better understand the design space for an intermediary architecture, we prototyped the intermediary architecture interceptor and support infrastructure and developed two services. Both were similarly structured.

- **Annotations Service** [Vasudevan and Palmer 1998a; 1998b] - This service permits anyone to author annotations to any document on the Web and for anyone else with the annotation viewer (implemented as an intermediary) to see these annotations. The process of authoring annotations can be supported in a wide variety of ways (our prototype supports several) and results in a collection of annotations of the form <URL - anchor point - annotation body - other metadata like author, data, security level, etc.>. These are stored in one or more metadata repositories, implemented via a DBMS. When a user who has installed the annotation viewer (implemented as an intermediary) browses the Web, not only is the page accessed but annotations from the annotation repository are also returned and composed with the page (again, several methods of composition are supported). If the annotation repository is local to a user or a workgroup, the user would see only local annotations; if it is public (for instance, implemented via a commonly available Web search engine), the user would see all (or a filtered set of) annotations. It is interesting to consider that the Annotation service effectively provides a way for third parties to augment web content, which provides a way for communities to share expertise. This kind of capability is useful in virtual offices like our own as well as in updating situation descriptions in military scenarios.
- **Personal Web Performance Monitor Service** [Hansen 1998a] - This service captures network performance metadata including trend data based on URL accesses. The user's browser issues a URL request and the monitor (implemented as an intermediary) intercepts it, pings the URL, and reports not only the accessed page but also the performance associated with the access. If the policy is to record trend data, the URL performance data is saved in a metadata repository, and accesses can result in not only current performance but historic performance. A straightforward extension would be to also use tracert to record the hops in the access, which can be useful when trying to figure out why a URL request is timing out - is it your ISP, the backbone, or the Web server?

4 Conclusions

Component-based software architectures are still immature. The Intermediary Architecture provides a productive architectural pattern for combining two of the most important mainstream component architecture frameworks, the Web and ORBs. It makes it easier for third-parties to produce plug-in services and opens the door a little wider for a component economy.

Based on our prototype work, we have learned some lessons and identified some open issues:

- One of our design goals was to avoid making changes to existing Web infrastructure and still install the IA architecture. We closely examined the pre-Mozilla Netscape browser client API to locate a hook for URL interceptors but found the API to be minimal and the then-available hooks to be insufficient. So in our current prototype, intermediaries are implemented as proxies. This allows IA to be portable across existing web infrastructure. Commercial browsers and servers might in the future incorporate IA capabilities to allow IA openness supported directly in clients or servers.
- The natural boundary of a service (component) is not always clear. For instance, should the performance monitoring service measure performance for the user-specified URL and each embedded URL (e.g., .gifs) separately or should it aggregate the result? The answer depends on what one is using the service for, which may not be entirely known at the time the service is defined. Transactional control boundaries are also an issue - to display a page, the Netscape browser client first issues a series of URL fetches including requesting embedded URLs but a proxy-based URL interceptor only sees a sequence of individual requests and not the fact that many are transactionally related to the page fetch.
- There appear to be many ways that services can interact with other services (including services of their own kind as in browser caching and proxy caching). Presently, scripting languages and system program languages are needed to compose services though we continue to look for stronger, more declarative results especially for important special cases like OMG basic object services. In addition, there is more to learn about composition of services. We have explored sequencing services in series but parallel and hybrid control structures would be useful too.
- Services *in the wild* are more difficult to tame than services in the lab. Knowing where to place intercept points is part of the problem; accessibility to make the connection is often lacking and undocumented; and obscure interactions can and often do occur. Wrapper approaches often replicate significant capabilities of the underlying system in order to install a new capability. Making use of undocumented APIs or formats, like accessing a file-based data structure, is subject to change in the underlying scheme.
- We have not yet considered in enough detail who can install new services in a user's path. IA seems to offer some nice opportunities for separation of roles via third party experts, but we have not examined IA security issues.

Acknowledgments

This research is sponsored by the Defense Advanced Research Projects Agency and managed by the U.S. Army Research Laboratory under contract DAAL01-95-C-0112. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied of the Defense Advanced Research Projects Agency, U.S. Army Research Laboratory, or the United States Government.

References

[Hansen 1998a]
HANSEN, G. Personal Web performance monitor service, OBJS Project Report (September 15, 1998), http://www.objs.com/OSA/Network-Performance-Monitor-Service.html

[Hansen 1998b]
HANSEN, G. Natural Language Query Interface, OBJS project report (September 15, 1998), http://www.objs.com/OSA/NLI-Query-Service.html

[Manola 1998a]
MANOLA, F. Towards a richer Web object model, *SIGMOD Record*, 27, 1 (March 1998), http://www.acm.org/sigmod/sigmod_record/

[Manola 1998b]
MANOLA, F. Technologies for a Web object model, Special issue on web object models, *IEEE Internet Computing*, (Jan/Feb 1999), http://www.objs.com/survey/wom-ieee.htm

[Pazandak 1998a]
PAZANDAK, P. Extending XML parsers to deserialize Java objects, OBJS technical report (September 15, 1998), To appear: http://www.objs.com/OSA/XML-to-Java-Mapping.html when released.

[Pazandak 1998b]
PAZANDAK, P. Intermediary Architecture Infrastructure, OBJS project report (September 15, 1998), http://www.objs.com/OSA/Intermediary-Architecture-Infrastructure.html

[Thompson 1998a]
THOMPSON, C. (ed), *OMG-DARPA Workshop on Compositional Architectures - Final Report*, Monterey, California (January 6-8, 1998). http://www.objs.com/workshops/ws9801/

[Thompson 1998b]
THOMPSON, C. Scaling object services architectures to the Internet, DARPA project report, (September 15, 1998), http://www.objs.com/OSA/Final-Report.html

[Thompson et. al. 1997]
THOMPSON, C., LINDEN, T., FILMAN, R. OMG OMA NG: Towards the next generation OMG object management architecture, OMG Document ormsc/97-09-01.html (October, 1997), http://www.objs.com/omg/OMG-OMA-NG.html

[Vasudevan 1998]
VASUDEVAN, V. A reference model for trader-based distributed systems architectures, OBJS technical report (January, 1998), http://www.objs.com/survey/trader-reference-model.html

[Vasudevan and Bannon 1998]
VASUDEVAN, V. and BANNON, T. Trader Service, OBJS project report (September 15, 1998), http://www.objs.com/OSA/Trader-Service.html

[Vasudevan and Palmer 1998a]
VASUDEVAN, V. and PALMER, M. Web Annotations Service, OBJS project report (September 15, 1998), http://www.objs.com/OSA/Annotations-Service.html

[Vasudevan and Palmer 1998b]
VASUDEVAN, V. and PALMER, M. Web annotation: promises and pitfalls of Web infrastructure, *32nd Hawaii International Conference on Systems Sciences* (January 1999), http://www.objs.com/survey/annotations-hicss.doc

[Wells et. al. 1992]
WELLS, D., BLAKELEY, J., THOMPSON, C. Architecture of an open object-oriented database management system." *IEEE Computer* (October 1992).

---

fax +1 (212) 869-0481, or permissions@acm.org.

Last modified: Saturday October 31 1998 23:00:00 CST